# データベース第10回

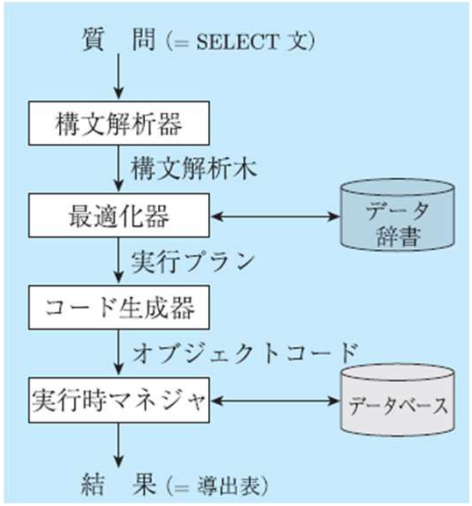
第10章　質問処理の最適化

## 質問処理の流れ

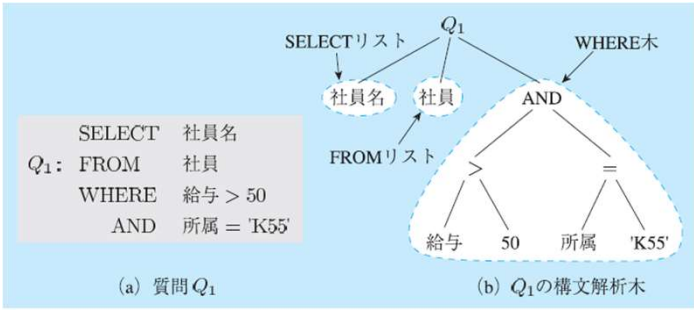
質問として与えられるSQL文は，以下の4段階に分けて処理される

1. 質問を構文解析する
2. 最適化を行う
3. 内部スキーマレベルのオブジェクトコードを生成する
4. オブジェクトコードを実行し結果リレーションを得る







# 単純質問の処理コスト

Q:
SELECT ＊
FROM R
WHERE A='a'

ただし，リレーションR(A,B)

- SQLの問いは非手続き的
- WHAT（何が欲しいのか）を表現
- HOW（所望のデータをどのような手続きで取ってくるか）は表現していない
- SQLの問いの処理の方法の工夫が重要



# Qの処理

1. インデックスは定義されていない．従ってRへのアクセスはスキャン
2. インデックスが定義されている．
   a. フィールドA上にインデックスXAがある
   b. フィールドB上にインデックスXBがある
   c. フィールドAとB上にそれぞれインデックスXAとXBがある



# Qの処理

1. インデックスは定義されていない．従ってRへのアクセスはスキャン

この処理コストは，

(Rのスキャン)+(A='a'の計算)



# Qの処理

2. インデックスが定義されている
   a. フィールドA上にインデックスXAがある

この処理コストは，

(XAを使ってA='a'のレコードの場所を探すコスト)
+(データをアクセスするコスト)
+(CPUコスト)

# Qの処理

2. インデックスが定義されている
   b. フィールドB上にインデックスXBがある

この処理コストは，

1. と同じ



# Qの処理

2. インデックスが定義されている
   c. フィールドA,B上にそれぞれインデックスXA, XBがある

この処理コストは，

今までの3種類の処理コストの一番少ない処理を選択できる

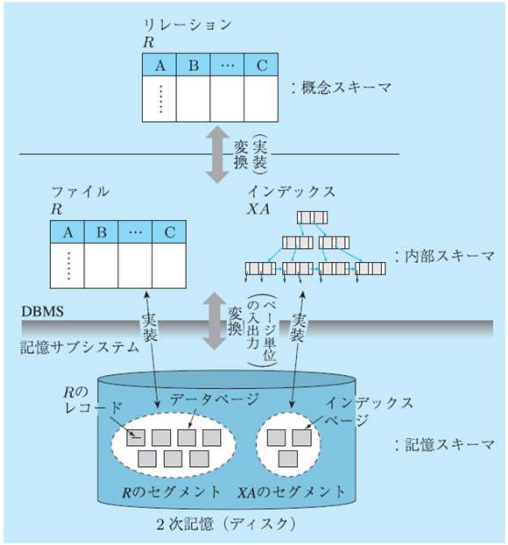


# Qの処理

1. インデックスは定義されていない．従ってRへのアクセスはスキャン
2. インデックスが定義されている．
   a. フィールドA上にインデックスXAがある
   b. フィールドB上にインデックスXBがある
   c. フィールドAとB上にそれぞれインデックスXAとXBがある

上記のそれぞれの場合のコストを見積もり，最適化を行う．その際，I/Oコストだけに注目し，CPU（計算）コストは無視することが多い．





# 結合質問の処理コスト

- 入れ子型ループ結合法
- ソートマージ結合法



# 入れ子型ループ結合

- R(A,B)とS(B,C)の自然結合R＊Sを考える

```
for each tuple r in R do
    for each tuple s in S do
        if r[B]=s[B] then output r＊s
    end
end
```

- RとSの順番に注意
- Sの結合属性上にインデックスが張られていると，処理効率は良くなる
- この場合のコストは
  (Rをアクセスするコスト)
  +N×(Sのインデックスを使ってアクセスするコスト)
  +(計算するコスト)
  ただし，NはRのタップルのうちSのタップルと結合可能なタップル数



# ソートマージ結合法

- R(A,B)とS(B,C)のソートマージ結合
  - 結合属性BでR,Sをソートしておく
  - Rで一つ，Sで二つのポインタを使って結合処理

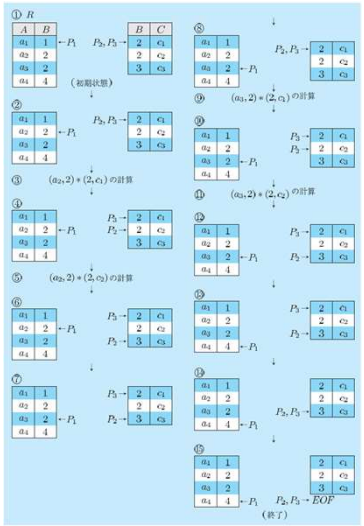
R

| A | B |
|---|---|
| a1 | 1 |
| a2 | 2 |
| a3 | 2 |
| a4 | 4 |

S

| B | C |
|---|---|
| 2 | c1 |
| 2 | c2 |
| 3 | c3 |





# ソートマージ結合法

- コストは以下となる

(Rをソートするコスト)
+(Sをソートするコスト)
+(R,Sをマージするコスト)
+(計算するコスト)

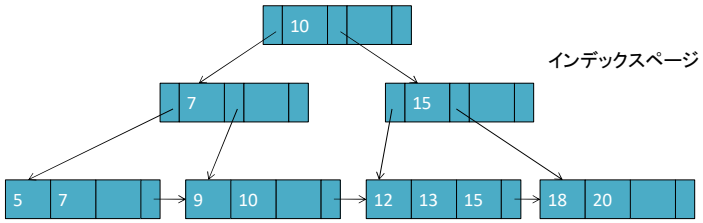
17

# 質問処理コストの推定

- 質問処理の最適化には，処理コストの見積もり(推定)が必要
- 推定に使える統計値(メタデータ)などが保存されている
  - TCARD(R):Rの大きさ(データ保存のブロック数やページ数)
  - NINDEX(XA):インデックスXAの大きさ(ブロック数やページ数)
  - ICARD(XA):Rの中の属性Aの異なった値の数(DISTINCTな数)
  - インデックスXAの性質
    - クラスタードインデックス
    - ノンクラスタードインデックス

18

# クラスタードインデックス

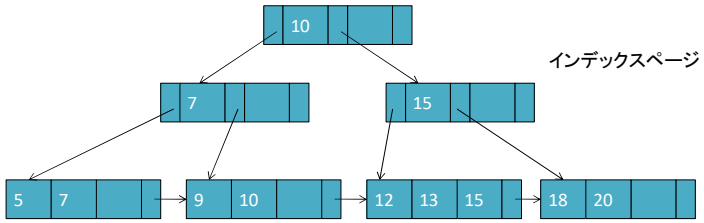



| | | | |
|---|---|---|---|
| 5; 5名前; 5年齢; 5職業 | 7; 7名前; 7年齢; 7職業 | 9; 9名前; 9年齢; 9職業 | 10; 10名前; 10年齢; 10職業 |
| 12; 12名前; 12年齢; 12職業 | 13; 13名前; 13年齢; 13職業 | 15; 15名前; 15年齢; 15職業 | 18; 18名前; 18年齢; 18職業 |
| 20; 20名前; 20年齢; 20職業 | | | |

データページ

19

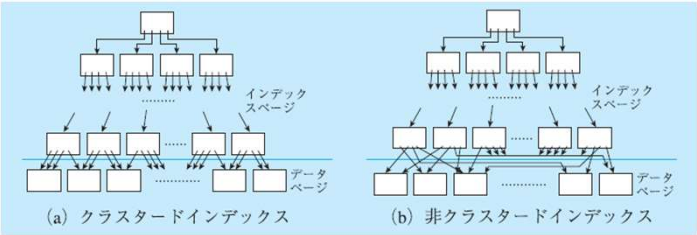
# ノンクラスタードインデックス




| | | | |
|---|---|---|---|
| 13; 13名前; 13年齢; 13職業 | 15; 15名前; 15年齢; 15職業 | 9; 9名前; 9年齢; 9職業 | 20; 20名前; 20年齢; 20職業 |
| 12; 12名前; 12年齢; 12職業 | 10; 10名前; 10年齢; 10職業 | 5; 5名前; 5年齢; 5職業 | 18; 18名前; 18年齢; 18職業 |
| 7; 7名前; 7年齢; 7職業 | | | |

データページ

20






# ヒューリスティックス

- R(A,B)とS(B,C)に関して

  SELECT ＊

  FROM R, S

  WHERE R.B=S.B AND R.A='a'



第11章　トランザクション

# トランザクション

- トランザクションとは，データベースに対するアプリケーションレベルでの一つの原子的な作用
  - 原子的(atomic)：これ以上分解できない
  - 作用(action)：データの読み(read)書き(write)
  - アプリケーションレベル：SQLの問合せ文と1対1に対応するわけではない＝いくつかの組合せ
- データベースに対してまとまって実行されるべき一連の作用の集まりをトランザクションという

# トランザクションの例

A氏の口座からB氏の口座へ100万円振替送金する

1. A氏の口座の残高が，100万円以上であることを確認．不足していたら終了．
2. A氏の口座の残高BalAから100万円を引き，BalA-100万に更新．
3. 続けて，B氏の口座の残高BalBに100万円を加え，BalB+100万に更新．

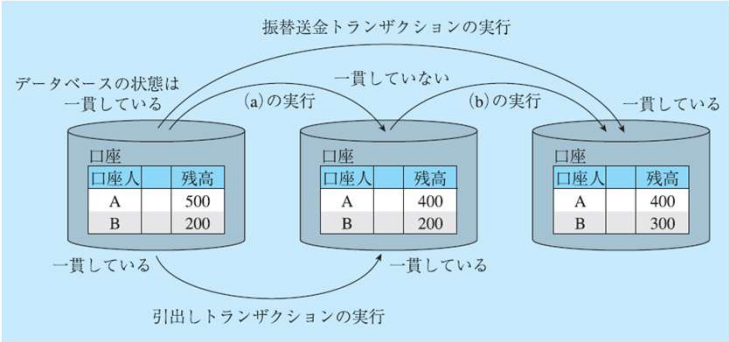
これら一連の作用は，まとまって処理されなければならない
- ステップ2が実行された後，ステップ3が処理されないとダメ
- ステップ2が実行されないのにステップ3が処理されるとダメ



# コミットとアボート

- トランザクションは，すべて正常に実行されるか，すべての実行が取り消されるか，いずれかでなければならない
  - 実行が完全に成功する：コミット(commit)
  - 取り消される：アボート(abort)もしくはロールバック(rollback)
- SQLでは，TRANSACTION BEGIN, COMMIT, ROLLBACKで指定する
- 何らかの原因によりトランザクションが正常に処理されなかった場合は，タイムアウトなどにより強制的にアボートする






# データベースの一貫性

- データベース＝実世界の写し絵
  - データベース内のデータの状態が，世の中の状態と矛盾してはならない
  - 社員が誕生日を迎えたら，社員リレーションの対応するデータの年齢も1増やす
- 100万円振替と，100万円引出すトランザクションを考えると．．．

# 引出すトランザクション

A氏の口座から100万円引出すことを考える

1. A氏の口座の残高が，100万円以上であることを確認．不足していたら終了
2. A氏の口座の残高BalAから100万円を引き，100万円の現金をATMで渡し，BalA-100万に更新



# 二つのトランザクション

**振替トランザクション**

1. A氏の口座の残高が，100万円以上であることを確認．不足していたら終了．
2. A氏の口座の残高BalAから100万円を引き，BalA-100万に更新．
3. 続けて，B氏の口座の残高BalBに100万円を加え，BalB+100万に更新．

**引出トランザクション**

1. A氏の口座の残高が，100万円以上であることを確認．不足していたら終了．
2. A氏の口座の残高BalAから100万円を引き，100万円の現金をATMで渡し，BalA-100万に更新．



```
TRANSFER: PROCEDURE OPTIONS (MAIN);  ← プログラムの開始
          DCL 依頼人 FIXED DEC (6,0);
          DCL 受取人 FIXED DEC (6,0);
          DCL 金額   FIXED DEC (9,0);
          DCL X      FIXED DEC (9,0);
          GET(依頼人, 受取人, 金額);
          EXEC SQL BEGIN TRANSACTION; ← トランザクションの開始
          EXEC SQL SELECT 残高
                   INTO X
                   FROM 口座
                   WHERE 口座番号 = 依頼人;
          IF X − 金額 < 0 THEN GO TO UNDO
          ELSE DO;
   (a)        EXEC SQL UPDATE 口座
                       SET 残高 = 残高 − 金額
                       WHERE 口座番号 = 依頼人;
   (b)        EXEC SQL UPDATE 口座
                       SET 残高 = 残高 + 金額
                       WHERE 口座番号 = 受取人;
              EXEC SQL COMMIT; ←トランザクションの終了（コミット）
              END;
          GO TO EXIT;
    UNDO: DO;
          PUT LIST ('残高不足');
          EXEC SQL ROLLBACK; ← トランザクションの終了（アボート）
          END;
    EXIT: END TRANSFER; プログラムの終了
```

```
WITHDRAWAL: PROCEDURE OPTIONS (MAIN);  ← プログラムの開始
          DCL 依頼人 FIXED DEC (6,0);
          DCL 金額   FIXED DEC (9,0);
          DCL X      FIXED DEC (9,0);
          GET(依頼人, 金額);
          EXEC SQL BEGIN TRANSACTION; ← トランザクションの開始
          EXEC SQL SELECT 残高
                   INTO X
                   FROM 口座
                   WHERE 口座番号 = 依頼人;
          IF X − 金額 < 0 THEN GO TO UNDO
          ELSE DO;
               EXEC SQL UPDATE 口座
                        SET 残高 = 残高 − 金額
                        WHERE 口座番号 = 依頼人;
               EXEC SQL COMMIT; ←トランザクションの終了（コミット）
               END;
          GO TO EXIT;
    UNDO: DO;
          PUT LIST ('残高不足');
          EXEC SQL ROLLBACK; ← トランザクションの終了（アボート）
          END;
    EXIT: END WITHDRAWAL; プログラムの終了
```

# 二つのトランザクション

**振替トランザクション**

1. A氏の口座の残高が，100万円以上であることを確認．不足していたら終了．
2. A氏の口座の残高BalAから100万円を引き，BalA-100万に更新．
3. 続けて，B氏の口座の残高BalBに100万円を加え，BalB+100万に更新．

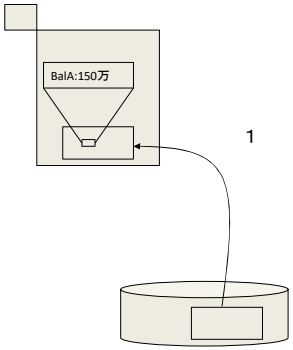
**引出トランザクション**

1. A氏の口座の残高が，100万円以上であることを確認．不足していたら終了．
2. A氏の口座の残高BalAから100万円を引き，100万円の現金をATMで渡し，BalA-100万に更新．

各トランザクションのステップ1，2を混ぜて実行すると両方で100万円(計200万円)が動く一方で，残高は100万円しか引かれないことになる

# 銀行口座からの現金引出し詳細

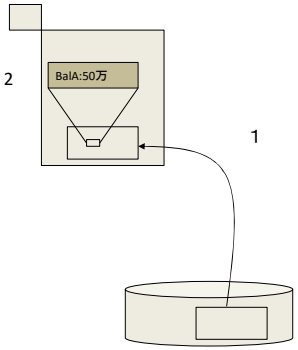
1. ディスク上のA氏の口座残高をメモリ上へ
2. 100万あるかどうかを確認し，BalA-100万をBalAに代入
3. 新しいBalAをディスクに書き込む




33

# 銀行口座からの現金引出し詳細

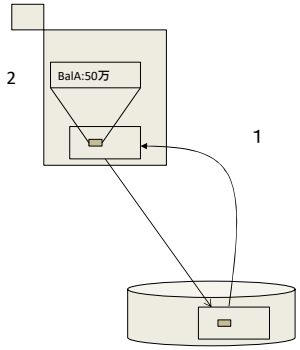
1. ディスク上のA氏の口座残高をメモリ上へ
2. 100万あるかどうかを確認し，BalA-100万をBalAに代入
3. 新しいBalAをディスクに書き込む



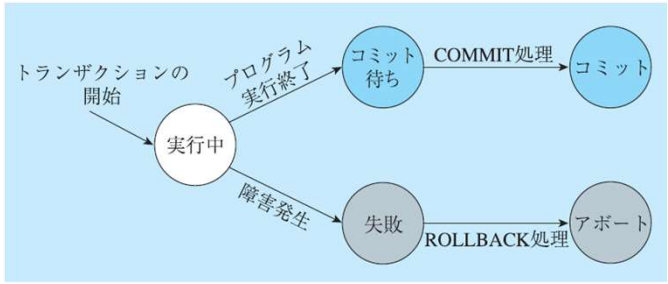

34

# 銀行口座からの現金引出し詳細

1. ディスク上のA氏の口座残高をメモリ上へ
2. 100万あるかどうかを確認し，BalA-100万をBalAに代入
3. 新しいBalAをディスクに書き込む




35



## ACID特性

- Atomicity, Consistency, Isolation, Durabilityの頭文字
- Atomicity：原子性
- Consistency：一貫性
- Isolation：隔離性，分離性
- Durability：耐久性，持続性

37

## Atomicity：原子性

- トランザクションは，それがすべて実行されるか，あるいは一切実行されないか，を保証しなければならない
- 全てが起こるか何も起こらないかのどちらか
- 途中で中断することは許されない
- 原子と同じで，それ以上分解できない性質をいう

38

## Consistency：一貫性

- トランザクションは状態を正しく遷移させる
- ひとまとまりの操作は当該状態に関する一貫性制約に違反することはない
- トランザクションは正しいプログラムであることが要求される

39

## Isolation：隔離性，分離性

- 複数のトランザクションが並行実行された場合にでも，それぞれのトランザクションTにとっては，他のトランザクションがTの前あるいは後のいずれかに実行されたように見え，Tの前後両方にまたがって見えたりしない
- 同時実行制御については第13章で

40

## Durability：耐久性，持続性

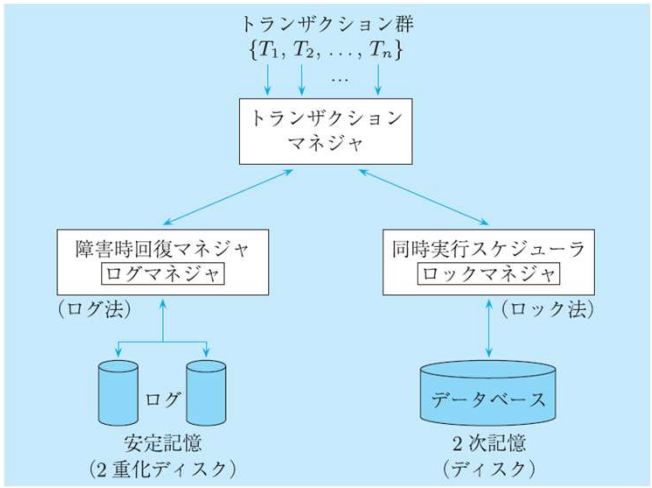
- 一度トランザクションが正常終了（コミット）すると，状態に対して行った変更は，いかなる障害が生じようとも持続する

41

## 例：銀行口座からの現金引出し

- 現金の引き渡しと口座の更新との両方がおこなわれれば，原子的である
- 渡された金額が口座の残高の減少分と同じ場合，一貫性がある
- 同時にこの口座を読み書きしている他のプログラムを気にしなくて良いなら，分離性がある
- 一度トランザクションが完了するとその口座の残高に引き出しが間違いなく反映されている場合，持続的である

42




43